

STP/HAMPI for Computer Security

Vijay Ganesh
vganesh@csail.mit.edu

March 2, 2013

Abstract

In the past several years I have written two SMT solvers called STP and HAMPI that have found widespread use in computer security research by leading groups in academia, industry and the government. In this note I summarize the features of STP/HAMPI that make them particularly suited for computer security research, and a brief description of some of the more important projects that use them.

1 Introduction

SMT solvers [3,4] (Satisfiability-Modulo-Theories Solvers) are computer programs that decide the satisfiability problem for rich logics such as the theory of bit-vectors and arrays [10], integers, and datatypes. SMT solvers have recently proven to be particularly useful in finding security vulnerabilities, debugging, and program analysis aimed at security. The reason for the success of SMT solvers are threefold: 1) The input logic of SMT solvers is rich enough to capture a wide variety of program behavior easily and compactly, 2) SMT solvers have become very efficient at solving such formulas obtained from real-world applications, and 3) there are very effective techniques now available, such as symbolic execution [7, 8, 11], that convert computation into SMT formulas. My solvers, STP [10] and HAMPI [12], are specifically designed to support computer security applications that perform security analysis aimed at finding security vulnerabilities [14], detecting malware [15] and constructing exploits [2, 6].

2 STP

STP [10] is a solver for a theory of bit-vectors and arrays. STP's logic is tailored to capture programs expressions exactly. All modern computer program expressions can be reduced to arithmetic and logic operations over suitably-sized (32 or 64 bit) bit-vectors or read/write operations over memory. STP's logic of bit-vectors captures program expressions, and STP's logic of arrays captures memory read/writes. This exact bit-precision allows users to easily encode a variety of security errors (e.g., off-by-one errors, memory errors, overflow errors).

STP has been used in more than 100 research projects, a good number of them are tools that automatically find security errors or perform binary analysis. Important examples include: BitBlaze project [15] from Dawn Song's group at Berkeley, The BAP system from David Brumley's group at CMU [5], EXE [8] and KLEE [7] from Dawson Engler's group at Stanford University, S2E project [9] from George Candea's group at EPFL, Switzerland, Akamai Inc. for finding security errors in mission critical applications (contact: Michael Stone), and governmental agencies. A comprehensive list of projects using STP can be found at the following website:

<http://sites.google.com/site/stpfastprover/tools-using-stp>

3 HAMPI

HAMPI [12] is a solver for a theory of strings that can solve constraints built out of string constants, variables, concatenation, extraction and membership in regular expressions and context-free grammars. HAMPI is explicitly aimed at finding security vulnerabilities, such as XSS attacks and SQL vulnerabilities, in web applications written in JavaScript, PHP and Python.

The big users of HAMPI include: The Ardilla tool [13] from Michael Ernst’s group at MIT and University of Washington Seattle, The WebBlaze project [14] from Dawn Song’s group at Berkeley, and Frank Tip’s group [1] at IBM T.J. Watson center at Hawthorne in New York.

A comprehensive list of all the tools using STP and HAMPI can be found by typing my name and following links at the Google Scholar’s page: <http://scholar.google.com>

References

- [1] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. Finding bugs in web applications using dynamic test generation and explicit state model checking. *IEEE Transactions on Software Engineering*, 36(4):474–494, July/August 2010.
- [2] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley. Aeg: Automatic exploit generation. In *Proceedings of the Network and Distributed System Security Symposium*, Feb. 2011.
- [3] C. Barrett, A. Stump, and C. Tinelli. The satisfiability modulo theories library and standard. <http://www.smtlib.org/>.
- [4] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
- [5] D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz. Bap: Binary analysis platform. In *Proceedings of the Computer Aided Verification Conference*, July 2011.
- [6] D. Brumley, P. Poosankam, D. Song, and J. Zheng. Automatic patch-based exploit generation is possible: Techniques and implications. In *Proceedings of the 29th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2008.
- [7] C. Cadar, D. Dunbar, and D. R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, 2008.
- [8] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: Automatically generating inputs of death. In *CCS*, 2006.
- [9] V. Chipounov, V. Kuznetsov, and G. Candea. The s2e platform: Design, implementation, and applications. *ACM Trans. Comput. Syst.*, 30(1):2, 2012.
- [10] V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In *CAV*, 2007.
- [11] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. In *PLDI*, 2005.
- [12] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: A solver for string constraints. In *ISSTA*, 2009.
- [13] A. Kiezun, P. J. Guo, K. Jayaraman, and M. D. Ernst. Automatic creation of SQL injection and cross-site scripting attacks. In *30th International Conference on Software Engineering (ICSE)*, May 2009.
- [14] P. Saxena, D. Akhawe, S. Hanna, S. McCamant, F. Mao, and D. Song. A symbolic execution framework for JavaScript. In *Accepted in the international proceedings of IEEE Security & Privacy*, May 2010.
- [15] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security, ICISS ’08*, pages 1–25, Berlin, Heidelberg, 2008. Springer-Verlag.